Herramienta de creación de problemas para jueces en línea

TRABAJO DE FIN DE GRADO

Gwydion J. Martín Ventura

Doble Grado en Ingeniería Informática y Matemáticas
Universidad Complutense de Madrid

Junio 2017

Documento maquetado con TEXIS v.1.0+.

Herramienta de creación de problemas para jueces en línea

Memoria que presenta para optar al título de
Doble Grado en Ingeniería Informática y Matemáticas

Gwydion J. Martín Ventura

Dirigida por los doctores

Marco Antonio Gómez Martín
Pedro Pablo Gómez Martín

Doble Grado en Ingeniería Informática y Matemáticas
Universidad Complutense de Madrid

Junio 2017

Resumen

En dos palabras puedo resumir cuanto he aprendido sobre la vida: sigue adelante.

Robert Lee Frost

Existen diversos jueces de programación en línea, plataformas que permiten a profesores y organizadores de concursos diseñar problemas y corregir de manera automática envíos que realicen los usuarios con posibles soluciones. Pero hay un inconveniente: cada juez se basa en un formato concreto de problemas, como veremos más adelante. Y por si fuera poco, además las herramientas que se facilitan para trabajar con los problemas son difíciles de utilizar, ya que suelen ser vía consola.

Por eso, en este proyecto se diseña una herramienta interactiva con interfaz gráfica que, partiendo del formato de problemas desarrollado en la Facultad de Informática, permite crear, modificar y validar problemas, mostrando algunas estadísticas y haciendo que el proceso sea mucho más sencillo de lo que era anteriormente. Se apoya en `ACREx`, un sistema implementado también en la Facultad que funcionaba con comandos de consola.

Palabras clave

Acepta el reto, juez en línea, formato de problemas, programación, Java, casos de prueba, interfaz, validación, soluciones.

Abstract

In three words I can sum up everything
I've learned about life: it goes on.

Robert Lee Frost

There are several online programming judges. They are platforms that allow teachers and organizers to create problems and automatically correct the sendings the users make with their own solutions. But there is a main problem here: each judge is based on a particular problem format, as we will see. Moreover, the tools to work with the problems are difficult to use: they are based on the command line.

This is the reason we decided to design an interactive, graphic tool, that uses the problem format developed in the Computer Science Faculty. It allows you to create, modify and validate problems, showing some statistics and making the process so much easy than it was before. It leans on `ACREx`, a system also developed in the Faculty that worked on the command line.

Keywords

Acepta el reto, online judge, problem's format, programming, Java, test-case, interface, validation, solutions.

Índice

Índice de figuras

Capítulo 1

Introducción

Lo último que uno sabe es por donde empezar.

Blaise Pascal

1.1. Motivación

Una rama importante de la programación es la referente a la programación de pequeños algoritmos que resuelvan diversos problemas, a la vez que deben ser óptimos en el uso del tiempo y la memoria. Se organizan muchos concursos de programación al año en diversas partes del mundo y con participaciones considerables, por lo que la corrección de los problemas uno a uno se hace impensable.

Por eso existen los jueces en línea, plataformas que, apoyadas en una base de datos de problemas de programación, permiten al usuario enviar soluciones que serán de manera automática corregidas y evaluadas, entrenándose así para futuros concursos a los que puedan presentarse (que, a su vez, se suelen organizar apoyados en estos mismos jueces). El principal "pero" de eso es que cada juez utiliza un formato diferente de problemas, como se verá más adelante. Esto supone que cuando un profesor u organizador de concurso quiere utilizar un juez nuevo, tiene que entender el funcionamiento del formato y luego aprender a manejar las herramientas que ofrezca el juez para trabajar con él. Por desgracia, muchas de estas herramientas se utilizan por consola, lo que no las hace intuitivas.

En la Facultad de Informática se desarrolló hace unos años *Acepta el reto*[1], un juez en línea con formato de problemas propio, además de las herramientas para trabajar con ellos, que forman `ACREx`. Pero éstas, de nuevo, consisten en comandos de consola. De ahí nace la idea de desarrollar una aplicación con interfaz gráfica que permita crear y modificar problemas, además

[1] `https://www.aceptaelreto.com`

de utilizar las herramientas que ya existen en `ACREx`.

1.2. ¿Por qué esto y no otra cosa?

En mi tercer año de carrera, allá por 2013, comencé a colaborar con Marco (uno de los directores del presente trabajo) y su departamento en la elaboración de problemas para *Acepta el reto*. Si bien la codificación de las soluciones y los generadores de prueba no era muy complicada, a la hora de preparar los archivos que formaban el problema como tal se pasaba un mal rato, ya que estaba compuesto por diversos directorios cuya información recogía un archivo `xml`: desde los autores del mismo hasta el número de soluciones o generadores que tenía, así como el lenguaje de cada uno de ellos. Entraremos más adelante a detallar el formato.

El problema, valga la redundancia, venía cuando querías añadir, por ejemplo, una nueva solución. Había que colocar los archivos fuente en una carpeta, esta dentro de otra carpeta de soluciones, y por último dentro de la raíz del problema. A su vez, había que modificar el archivo `xml` para que recogiese esta nueva alternativa. Y así con cualquier mínimo cambio que se quisiera realizar.

Y cuando una persona nueva se incorporaba al equipo de preparación de problemas, debía entender todo el esquema de los mismos. En resumen, que además de ser buen programador, había que saber XML y dominar la estructura que se utilizaba. Y todo esto lo sufrí yo mismo, cuando apenas tenía conocimientos de programación y quise colaborar. Así que cuando se me propuso diseñar una interfaz gráfica que controlase toda la generación y modificación de problemas, que liberase al usuario de trabajar con el archivo `xml` raíz y que simplificase al máximo todo el trabajo, no me lo pensé dos veces.

1.3. Esquema de la memoria

En el capítulo 2, estudiaremos diversos jueces en línea y formatos de problema utilizados en otras instituciones o países.

En el capítulo 3, analizaremos *Acepta el reto*, el juez en línea de la Facultad. También veremos qué herramientas ofrecía `ACREx`, pues es el pilar en el que se apoya esta aplicación.

En el capítulo 4, describiremos la interfaz de la aplicación, además de cómo utilizarla y los resultados que obtenemos después de validar un problema.

En el capítulo 5, veremos cómo se ha implementado la aplicación, incluyendo las decisiones de diseño que se han tenido que tomar, y describiendo las librerías externas utilizadas.

Por último, en el capítulo 6 se exponen las conclusiones finales y se proponen posibles cambios futuros que se pueden realizar sobre la aplicación.

Capítulo 2

Estado del arte

En lo que parecemos, todos tenemos un juez; en lo que somos, nadie nos juzga.

Friedrich Schiller

Resumen: en este capítulo, se estudian diversos jueces en línea, españoles y extranjeros, viendo qué ofrece cada uno de ellos, en qué se diferencian y cuáles son sus semejanzas. A continuación, se analizan dos formatos de problemas ampliamente utilizados en concursos internacionales.

2.1. Jueces en línea

Los jueces en línea son herramientas que permiten a los programadores entrenarse resolviendo diversos problemas que hay almacenados en la base de datos del mismo, que posteriormente envían al juez y éste los evalúa. También se utilizan a la hora de realizar concursos de programación, ya que facilitan la tarea de los organizadores para obtener los resultados de los distintos concursantes.

En los siguientes apartados, veremos cuatro de ellos: dos de origen español, procedentes de las universidades de Valladolid y Politécnica de Cataluña, y dos del este de Europa, concretamente de Polonia y Rusia.

2.1.1. UVa Online Judge

La Universidad de Valladolid cuenta con un juez en línea creado en 1995 por Miguel Ángel Revilla, profesor de matemáticas y algoritmia, y Ciriaco García de Celis, uno de sus estudiantes. Se hizo público dos años más tarde, y en 1999 fueron sede de las *SWERC* (Southwestern Europe Regional Contest),

uno de los concursos de programación más prestigiosos a nivel europeo. El juez actual[1] fue desarrollado en 2007, reemplazando el anterior.

Llama la atención que toda la web del juez esté en inglés, siendo una plataforma de origen español. Sin embargo, la interfaz es lo suficientemente clara como para entenderla sin problemas.

Contiene más de 3000 problemas, que se pueden encontrar clasificados por categorías, como interactivos o de concursos, así como por volúmenes organizador por fecha de creación. Cuando accedemos a la ficha de un problema concreto, se nos muestra el enunciado (que incluye la motivación y el formato de entrada y salida, con ejemplos incluidos), junto con su límite de tiempo, además de botones para enviar una solución, depurarla, mostrar estadísticas o descargar el PDF con el enunciado. Además, antes de entrar a un problema, en la vista de una categoría o volumen podemos ver el porcentaje de aciertos respecto de los envíos y de los usuarios.

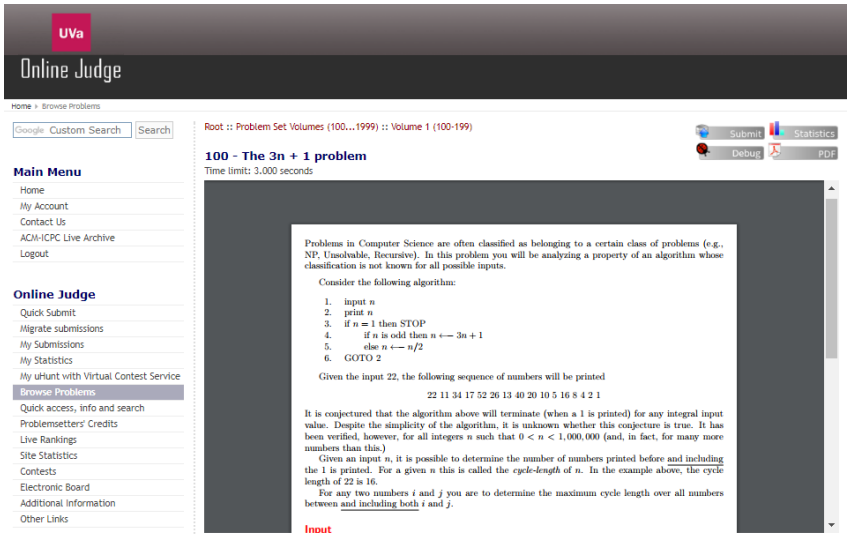


Figura 2.1: Vista de un problema en UVaOJ

Una de las funcionalidades más útiles de UVaOJ es la de *Quick Submit* o "envío rápido", que simplemente indicando el ID del problema a resolver y el lenguaje en el que está programado, y proporcionando el código bien subiendo un archivo fuente o bien pegando el código en un área de texto, se realiza el envío, sin tener que acceder a la ficha del problema concreto.

Por último, podemos destacar *My uHunt*, una herramienta desarrollada por Felix Halim en la Universidad de Singapur, que se encarga de registrar estadísticas y proponer problemas a los usuarios en base a los envíos que han realizado anteriormente. No es propia de la UVa, pero está adaptada a la base de datos del juez. También implementa un chat en el que los usuarios

[1] `http://uva.onlinejudge.org`

se comunican para resolver dudas, además de estar disponible la API para obtener estadísticas en vivo acerca del juez online.

2.1.2. Jutge

Jutge[2] es el juez en línea de la Universidad Politécnica de Cataluña, desarrollado por Jordi Petit y Salvador Roura, ambos profesores del Departamento de Ciencias de la Computación. Es el juez utilizado en la Olimpiada Informática Española, lo que muestra su robustez y completitud.

Contiene un amplio catálogo de problemas, aunque la mayoría de ellos están en catalán o inglés, así como una sección de cursos que engloban diversos problemas. Llama la atención el curso "Learning to program": programas sencillos para aprender a programar, de manera que comienza por tareas sencillas (operaciones numéricas, bucles) y va subiendo de nivel progresivamente (procedimientos, recursión, vectores...).

Soporta multitud de lenguajes, desde los más comunes como Java, C++ o Python, hasta los más inesperados, como Brainfuck, Lua o Whitespace. Además, tiene un sistema de logros, que se van concediendo según resuelves problemas.

Por último, Jutge permite descargar certificados de envíos corregidos, incluyendo en un archivo comprimido firmado la solución enviada e información sobre el autor de la misma.

2.1.3. Sphere

Esta plataforma[3] polaca, propiedad de Sphere Research Labs, es una de las más utilizadas en el mundo. Se ha utilizado en más de 2400 concursos en los últimos 5 años, pues la creación de éstos es muy sencilla y rápida.

Tiene aproximadamente 13.000 problemas, que pueden ser resueltos en más de 45 lenguajes de programación diferentes. En este caso, y en esto mejora a Jutge, implementa un editor de textos en la propia web, que además incluye una plantilla dependiendo del lenguaje de programación que seleccionemos.

Tiene también un "Hall de la fama" en el que se aparecen los mejores resolutores de problemas que hay actualmente registrados, y una sección Status que muestra las tareas que está realizando el juez en ese mismo momento (ver figura 2.3).

[2] `http::://jutge.org`
[3] `http://www.spoj.com/`

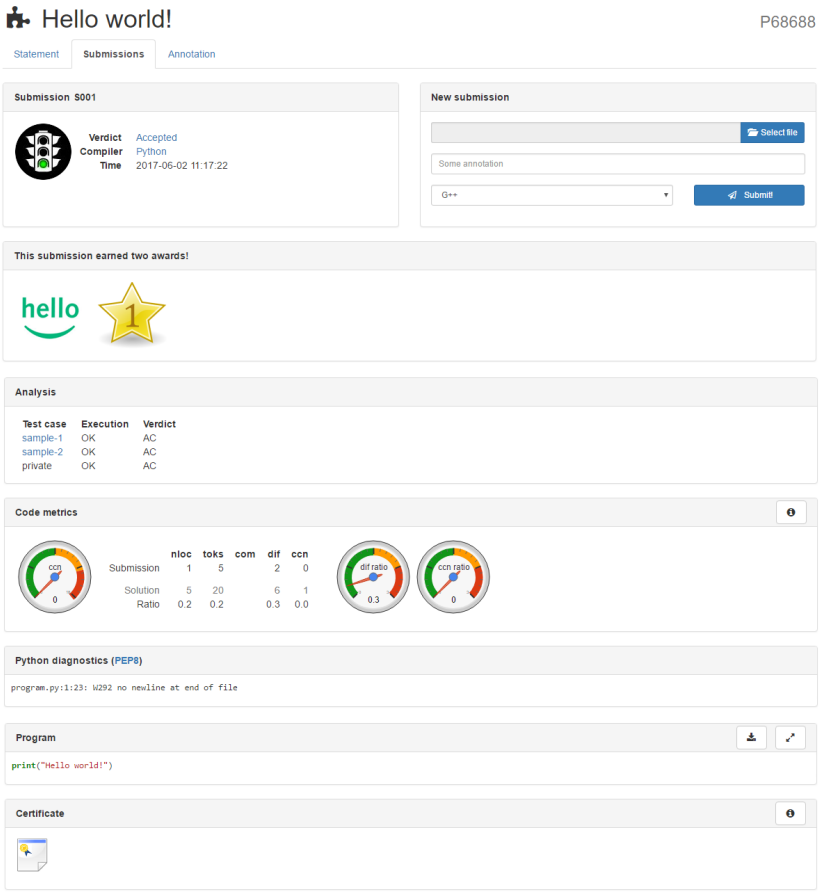


Figura 2.2: Resultado de una entrega en Jutge

2.1.4. CodeForces

De Polonia nos vamos a Rusia, donde nació CodeForces[4] en 2010. De los cuatro jueces que hemos analizado, éste es el que cuenta con una interfaz más sencilla e intuitiva, pero no por ello es el más sencillo de todos.

Además del gran banco de problemas que almacena, y que se pueden enviar en cualquier momento, cuenta con dos modos competitivos: los concursos (*Contest*) y los entrenamientos (*Gym*). Si bien ambos tienen una fecha y duración determinadas, la diferencia principal radica en que en los concursos se pueden pasar algunos tests de manera previa a enviar la solución, pero a la hora de corregirlo se utilizarán otros, presumiblemente más complicados; en cambio, en el modo de entrenamiento se pueden ejecutar todos los tests disponibles para un problema, además de recibir comentarios sobre el resultado.

[4] `http://codeforces.com/`

Figura 2.3: *Status* de Sphere

Otra de las utilidades que incluye es un sistema de grupos privados, los cuales pueden organizar sus propios concursos internos y prepararse de cara a futuros eventos globales. En la barra de herramientas de la web aparecen también los grandes concursos, como puede ser la Copa Rusa de Código.

Pero lo que realmente diferencia a CodeForces del resto de jueces analizados es la API propia que ofrecen, con la que podemos acceder a parte de sus datos en formato JSONECMA-404 (1999). Mediante peticiones HTTP a la dirección `http://codeforces.com/api/{nombreDelMétodo}`, se recibe la información requerida. Entre los métodos, se encuentran `blogEntry.comments` para ver los comentarios de una entrada concreta, `contest.ratingChanges` para conocer qué usuarios han conseguido puntos después de un concurso o `user.status` para saber cuáles son los envíos que ha realizado el usuario en cuestión.

2.2. Formatos de problemas

Cuando hablamos de un problema de programación, obviamente necesita un enunciado que plantee la tarea a realizar. Sin embargo, al querer introducirlo en un juez en línea, necesita de más elementos que permitan evaluar la entrega del usuario, como pueden ser una solución válida, unos casos de prueba, unos ejemplos de entrada/salida, etc. En los siguientes apartados, veremos diferentes formatos de problemas que se pueden encontrar en la web.

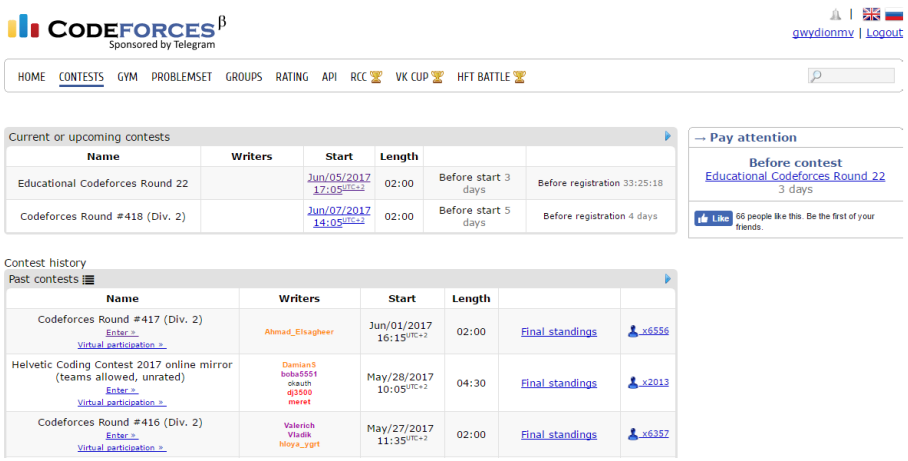


Figura 2.4: Concursos en CodeForces

2.2.1. Polygon

Polygon[5] es una plataforma online (en fase beta) desarrollada en Rusia que ofrece "un método profesional de preparar problemas para concursos de programación". Cualquiera puede acceder a ella, sólo es necesario crear una cuenta de usuario gratuita.

Una vez creada, se ofrece una interfaz sencilla pero clara para crear los problemas, separando cada parte de los mismos en diferentes páginas. Por ejemplo, en una se indica la información general, como los archivos de entrada y salida, los límites de tiempo y memoria o las etiquetas del problema; en otra, se diseña el enunciado en el idioma deseado, introduciendo cada uno de los apartados en campos concretos (leyenda, formatos de entrada y salida, notas o tutorial).

Ofrece multitud de funcionalidades: comprobación de soluciones por comparación con una solución correcta, validación del problema respecto a los tests, inclusión de paquetes predefinidos y un panel de mensajes, que soporta propuestas de mejora, informe de errores y discusiones entre usuarios. Además, permite crear concursos de programación a partir de los problemas ya creados, recopilando toda su información y permitiendo el acceso a otros usuarios.

Por último, me gustaría destacar los siguientes aspectos de Polygon:

- Con sólo un click, muestra el enunciado en LaTeX, HTML y PDF a partir de los datos introducidos anteriormente.
- Incluye una gestión del acceso de los usuarios al problema, que permite conceder permisos de lectura, escritura o ninguno a cada uno de ellos.

[5] `https://polygon.codeforces.com`

- Implementa *edit sessions* (sesiones de edición), que permite trabajar con copias locales en el servidor. Esto implica que si tenemos cualquier problema y perdemos la conexión a internet o se apaga nuestro equipo, la copia quedará guardada en el servidor, pero sin mostrar los cambios al resto de usuarios hasta que no hagamos un *commit*.
- Relacionado con los *commits*, no implementa resolución de conflictos; es decir, que si se realizan modificaciones por dos usuarios y hay conflictos, una de las dos revisiones se destruye.

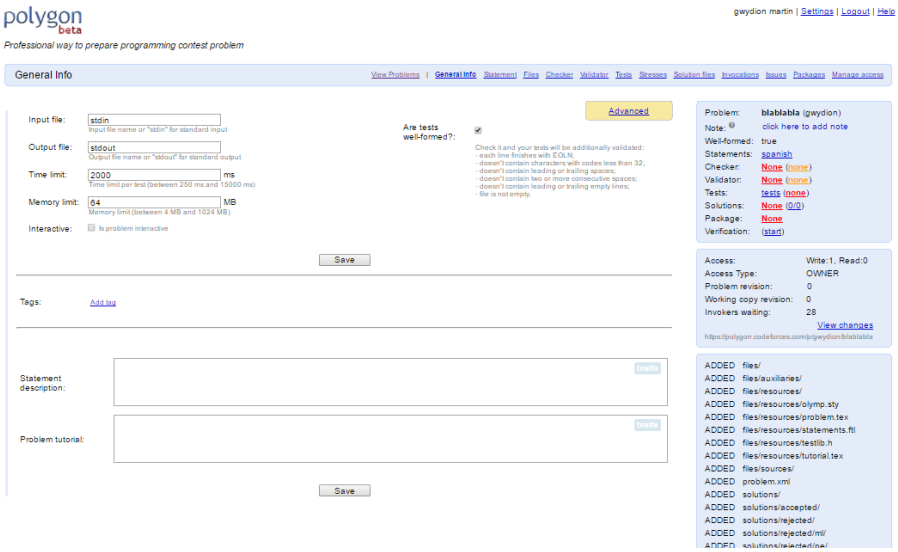


Figura 2.5: Vista "Información general" de un problema en Polygon

2.2.2. Kattis

Otro formato utilizado en la web es Kattis[6], creado en 2003 por Pehr Söderman junto con la ayuda de cuatro compañeros suyos del Real Instituto de Tecnología (Kungliga Tekniska Högskolan, KTH) de Estocolmo (Suecia). Sin embargo, se utilizó exclusivamente de manera privada en actividades de la universidad, hasta que se creó OpenKattis[7] en junio de 2013.

Se basa en un sistema de directorios para cada problema y un archivo de extensión YAML que contiene la información básica del mismo (fuente, autor, licencia del problema, límites de ejecución e información para la validación). Como vemos en la figura 2.6, es una simple relación entre los diferentes atributos y su valor.

El diseño del directorio es el siguiente:

[6] `http://problemarchive.com`
[7] `http://open.kattis.com`

```
source: Nordic Collegiate Programming Contest 2015
source_url: http://ncpc.idi.ntnu.no/
author: Lukáš Poláček
# rights_owner:
license: cc by-sa

#limits:
#    time_multiplier: 5
#    time_safety_margin: 2
#    memory: 4096
#    output: 16
#    compilation_time: 240
#    validation_time: 240
#    validation_memory: 3072
#    validation_output: 4
#validator_flags: space_change_sensitive float_absolute_tolerance 1e-6
validation: custom
```

Figura 2.6: Ejemplo de archivo `problem.yaml`

- *Datos de prueba:* todos en la carpeta `/data` pero separados en dos directorios, uno para los que se incluyen en el enunciado (`/sample`) y otro para el resto de ejemplos que se probarán en los tests (`/secret`).
- *Enunciado:* carpeta `/problem_statement` con el archivo .tex que describe el problema, y el resto de archivos necesarios para su compilación. Es importante resaltar que sólo admite un enunciado, por lo que no soporta traducciones.
- *Validadores:* se incluyen dos carpetas, una para validar la entrada (`/input_format_validators`) y otra para la salida (`/output_validators`), aunque ésta no se suele utilizar.
- *Entregas:* en la carpeta `/submissions` se recogen programas válidos (`/accepted`), incorrectos (`/wrong_answer`), erróneos (`/run_time_error`) y que se exceden en tiempo de ejecución (`/time_limit_exceeded`).

Todos los elementos, a excepción de los validadores de salida, son necesarios para que el problema se considere correctamente formateado.

Además, Kattis ofrece tres herramientas para trabajar con los problemas en el paquete `problemtools`:

- `verifyproblem`, que ejecuta una comprobación general de todo el problema.
- `problem2pdf`, que genera un pdf con el enunciado del problema.
- `problem2html`, que genera un html con el enunciado del problema.

Capítulo 3

Punto de partida

Toda la gloria proviene de atreverse a comenzar

Eugene F. Ware

Resumen: en este capítulo, veremos qué es *Acepta el reto*, el juez en línea para el que está pensada la aplicación desarrollada, algunas estadísticas de la plataforma y todo lo que ofrece a usuarios. También analizaremos el formato de problemas que se utiliza, ya que es el mismo con el que trabajaremos posteriormente en la aplicación. Por último, se estudia el conjunto de herramientas que ofrece el paquete `ACREx`.

3.1. *Acepta el reto*

Como se ha mencionado anteriormente, *Acepta el reto* es una plataforma web en la que se plantean numerosos problemas de programación, propios o inspirados en otros que han aparecido en concursos de todo el mundo, y se ofrece al usuario la posibilidad de resolverlos (en C, C++ o Java) y enviar la solución para su evaluación.

Los creadores son Marco Antonio y Pedro Pablo Gómez Martín, ambos co-directores de este trabajo e integrantes del Grupo de Aplicaciones de Inteligencia Artifical (GAIA)[1]. Actualmente, cuenta con más de 300 problemas distribuidos de dos formas diferentes: por un lado, en volúmenes de 100 problemas según su ID; por otro, en categorías (programación, concursos, exámenes, temática... y sus propias subcategorías). Además, en la página principal se destaca el "problema de la semana".

[1] `http://gaia.fdi.ucm.es`

Figura 3.1: Vista de la portada de *Acepta el reto*

La evaluación de los envíos se realiza mediante la ejecución de distintos casos de prueba específicos para cada problema y la posterior comprobación de la salida obtenida con la esperada. Los posibles resultados son:

- *Accepted (AC)*: la solución es válida.
- *Presentation Error (PE)*: la ejecución es correcta, pero hay errores en saltos de línea, espacios o tabuladores.
- *Wrong Answer (WA)*: se han cometido errores en la ejecución de los casos de prueba.
- *Compilation Error (CE)*: el código enviado no compila.
- *Run-time Error (RTE)*: hay un fallo en la ejecución del código que provoca un final inesperado.
- *Time Limit Exceeded (TLE)*: la ejecución excede el tiempo límite marcado por el autor del problema.
- *Memory Limit Exceeded (MLE)*: la ejecución sobrepasa el límite de memoria establecido.
- *Output Limit Exceeded (OLE)*: se escribe más salida de la permitida.
- *Restricted Function (RF)*: se detiene la ejecución del programa debido a que se considera que una de las operaciones a realizar era ofensiva o peligrosa para el servidor.
- *Internal Error (IE)*: el servidor ha sufrido un problema al ejecutar el código, por lo que es necesario un reenvío para que se vuelva a probar.

Como podemos ver en la figura 3.2, se realizan envíos muy frecuentemente a la plataforma; en este caso, son cuatro los usuarios que están conectados, y todos están programando en Java. Un vistazo rápido a los últimos 80 envíos nos dice que sólo cuatro de ellos fueron en C++, todos del mismo usuario y sobre el mismo problema. En el caso de los envíos correctos, se informa del tiempo y la memoria consumidos, así como de la posición en el ranking particular de ese problema.

Envío	Usuario	Problema	Resultado	Lenguaje	Tiempo	Memoria	Pos	Fecha
139729	primagaz	Las perlas de la condesa (3...	RTE	Java	-	-	-	Hace 5 minutos
139728	primagaz	Las perlas de la condesa (3...	RTE	Java	-	-	-	Hace 12 minutos
139727	EmX_Fyxed1	Repartiendo regalos en tu ...	AC	Java	0.251	1644	12	Hace 13 minutos
139726	EmX_Fyxed1	Repartiendo regalos en tu ...	PE	Java	0.239	1711	-	Hace 20 minutos
139725	primagaz	Las perlas de la condesa (3...	RTE	Java	-	-	-	Hace 24 minutos
139724	xabier2012	Tira y afloja (336)	MLE	Java	-	-	-	Hace 36 minutos
139723	adaljj96	Distancia al siguiente capic...	RTE	Java	-	-	-	Hace 38 minutos
139722	arthur170707	Ventas (105)	RTE	Java	-	-	-	Hace 40 minutos
139721	arthur170707	Ventas (105)	RTE	Java	-	-	-	Hace 41 minutos
139720	xabier2012	Tira y afloja (336)	RTE	Java	-	-	-	Hace 43 minutos
139719	jjurrea	Alan Smithee (260)	AC	Java	3.697	1754	77	Hace 1 hora
139718	jjurrea	Alan Smithee (260)	AC	Java	3.932	1754	80	Hace 1 hora

Figura 3.2: Vista de los últimos envíos en *Acepta el reto*

3.2. Formato de problemas de *Acepta el reto*

Cuando el usuario se enfrenta a un problema de programación en *Acepta el reto*, sólo ve el enunciado del mismo, que incluye el título, la descripción y el formato de la entrada y la salida, incluyendo un ejemplo formado por una entrada y su salida esperada. Sin embargo, como el juez debe ser capaz de corregir las soluciones, crear un problema no es simplemente escribir su enunciado: hacen falta bastantes cosas más.

El formato con el que vamos a trabajar consiste en que cada problema va a estar formado por un archivo `xml` y cuatro directorios, que incluirán a su vez diversos archivos.

- `problem.xml`: es el archivo raíz que condensa toda la información del problema. Contiene información de los autores y comentarios, así como referencia a todos los archivos que conforman el problema (soluciones, generadores, enunciados...).

- `/sample`: contiene dos archivos de texto con ejemplos de la entrada (`empty.in`) y la salida (`empty.out`).

- `/solutions`: contiene los archivos de código fuente de las soluciones programadas por los autores, separados entre aquellos correspondientes a la solución oficial (en la carpeta `/official`) y al resto (dentro de la carpeta `/other` hay otra para cada solución).
- `/statements`: se incluyen los enunciados del problema en archivos `xml`; en plural, ya que admite diversos idiomas, cada uno en un archivo y en la carpeta correspondiente a su idioma (el enunciado en español, por ejemplo, será `/statements/Spanish/statement.xml`).
- `/testcases`: por un lado, están los archivos `empty.in` y `empty.out`, que representan la entrada y la salida vacías del problema; por otro, la carpeta `/generators`, que incluye archivos `testcases#` con la codificación de los generadores de los casos de prueba (`#` será el ID del generador).

3.3. ACREx

El proyecto `ACREx`, programado en Java, cuenta con nada menos que 115 archivos de código fuente. Permite cargar problemas (que deben estar en el formato descrito en la sección anterior), validarlos (comprobando, por ejemplo, que cada archivo está en la carpeta correspondiente), compilar y ejecutar las soluciones para ver si en verdad lo son y obtener el enunciado en PDF a partir del `xml` en el que se escribe.

Consta de multitud de paquetes, todos dentro del paquete base `es.acrex`, entre los cuales destacan los siguientes:

- `comparator`: incluye las clases encargadas de realizar la comprobación de la salida de las soluciones.
- `compiler`: engloba las clases que representan un compilador de los lenguajes admitidos, Java, C y C++, así como de LaTeXpara la generación de los enunciados.
- `tools`: paquete muy variado, con clases para comprobar soluciones, compilar enunciados, obtener listas de soluciones o enunciados o comprimir archivos. También incluye la herramienta utilizada como validador de consola antes del desarrollo de la aplicación.
- `utils`: incluye clases para trabajar con flujos de entrada/salida o con archivos, como copiarlos o deducir el lenguaje en el que están programados, con o para aspectos del sistema operativo, así como para conocer el sistema operativo en el que se está ejecutando la aplicación.
- `validator`: es el encargado de realizar las comprobaciones necesarias para validar el problema, al igual que de informar de los resultados obtenidos.

- `xml`: contiene las representaciones de los datos del problema que se incluyen en el archivo `problem.xml`, como pueden ser las personas (autores o revisores), las soluciones o los enunciados. Éstas son las representaciones encargadas de leer de XML para cargar un problema y de escribir en XML para guardarlo.
- `xslt`: realiza la transformación del archivo `xml` del enunciado para obtener el `tex` que se compilará posteriormente.

Además, en el paquete raíz `es.acrex` también hay varias clases, que representan los datos con los que se trabaja en la aplicación: personas, problemas, soluciones, enunciados...

Capítulo 4

Manual del usuario

Uno se alegra de resultar útil

El hombre bicentenario

Resumen: en este capítulo, explicaremos el funcionamiento de la herramienta, las distintas opciones que ofrece y los resultados que podemos obtener de ella.

4.1. Inicio de la aplicación

Cuando arrancamos la herramienta, lo primero que se nos muestra es la ventana inicial (ver figura 4.1), donde se presenta el nombre de la misma y se nos da opción a crear un problema desde cero o a cargar uno ya existente.

4.1.1. Creación

Cuando elegimos crear un problema nuevo, la aplicación solicita el nombre que queremos darle, así como el directorio en el que queremos que se guarde.

Se monta el esqueleto de un problema, es decir, las carpetas `/sample`, `/solutions`, `/statements` y `/testcases` dentro de la carpeta raíz que hayamos indicado anteriormente. Además, se copiará al directorio un `problem.xml` vacío, en el que se guardarán los cambios conforme se vayan realizando en el futuro.

Una vez hecho esto, se muestra la ventana principal, que describiremos en la siguiente sección.

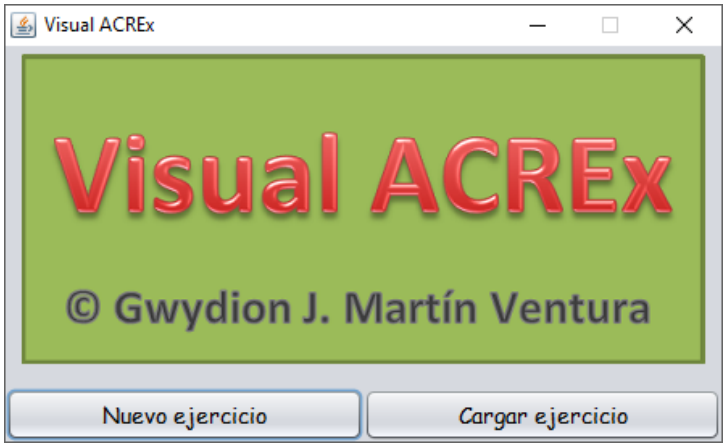


Figura 4.1: Ventana inicial

4.1.2. Modificación

Si decidimos modificar un archivo ya existente, podemos elegir entre cargar un archivo `zip` con el problema o leer un directorio en el que se encuentre. Es claro que el problema debe encontrarse en el formado descrito en la sección 3.2, ya que si no es así el programa no será capaz de leerlo correctamente y tendremos un fallo de carga. Si todo va bien, se nos muestra la ventana principal, que pasamos a describir a continuación.

4.2. Ventana principal

La venta está estructurada en varias pestañas, de manera que podemos acceder a la vista del campo que queramos sin que nos moleste el resto de información. Por defecto, cuando se abre se muestra la pestaña de *Metadatos*. Junto a ésta, tenemos las pestañas de *Enunciado*, *Soluciones y Generadores*, *Ejemplos* y *Archivos*, que veremos detenidamente en las siguientes subsecciones.

Además, en la parte superior de la ventana principal se incluyen dos botones: uno permite guardar los cambios realizados, y el otro validar el problema. Es importante resaltar que, cuando se hace click en *Validar*, se guarda automáticamente, ya que no tiene sentido realizar cambios en un problema y luego intentar validar la versión anterior.

4.2.1. Metadatos

En esta primera pestaña se incluyen los datos generales del problema, como son el nombre y la ruta donde se encuentra, además de una tabla con los autores que han participado en la elaboración del problema. De cada uno,

se incluye su nombre y su correo electrónico. Se da opción a añadir autores nuevos o a eliminarlos.

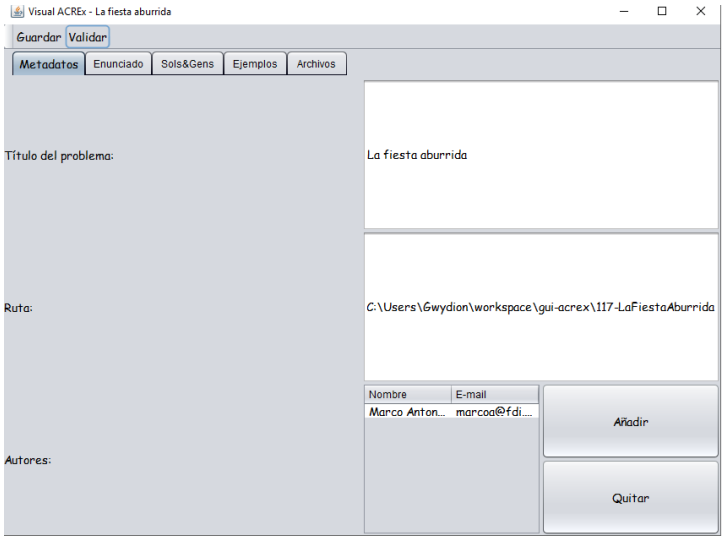
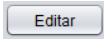
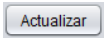


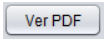
Figura 4.2: Vista de la pestaña *Metadatos*

4.2.2. Enunciado

Se muestra el XML que incluye toda la información del enunciado, es decir, el título del mismo, la descripción/motivación general, el formato de la entrada y la salida del problema, pistas para los alumnos e información para el profesor. Junto a él, aparecen tres botones:

- 

: abre el archivo del enunciado en el editor de textos por defecto del usuario.
- 

: pensado para refrescar el área de texto que muestra el enunciado después de haber realizado cambios en el mismo.
- 

: abre el archivo PDF compilado a partir del XML; de nuevo, utilizando el visor por defecto que haya configurado el usuario.

4.2.3. Soluciones y generadores

Esta pestaña muestra dos tablas, y los botones correspondientes para cada una de ellas: en la parte superior, se encuentra la información correspondiente a las soluciones del problema, y debajo, la de los generadores de casos de prueba.

- La tabla para las soluciones muestra el autor, el lenguaje en el que está programada y el veredicto de la solución. Además, se indica cuál de ellas es la solución oficial, que siempre debe estar programada en C o C++. Junto a ella, tres botones: añadir, eliminar y seleccionar como oficial.

- La tabla para los generadores incluye el ID del generador, el autor y el lenguaje en que se programa. En este caso, sólo dos botones: añadir y eliminar.

Cuando añadimos una solución, se solicita cada uno de los campos mencionados. Si el autor no aparece en la tabla de autores de la pestaña *Metadatos*, se abrirá una nueva ventana para crear un autor nuevo. De esta manera, no aseguramos que todo autor que queramos introducir aparezca como autor del problema en el campo general.

Es importante resaltar que cuando se elimina una solución o un generador, en realidad simplemente se está eliminando de la representación del problema que hay internamente en la aplicación. Así, hasta que no se guarden los cambios, los archivos siguen formando parte del problema, y una vez que se guarden, pasarán a formar parte de la pestaña *Archivos* (ver sección 4.2.5).

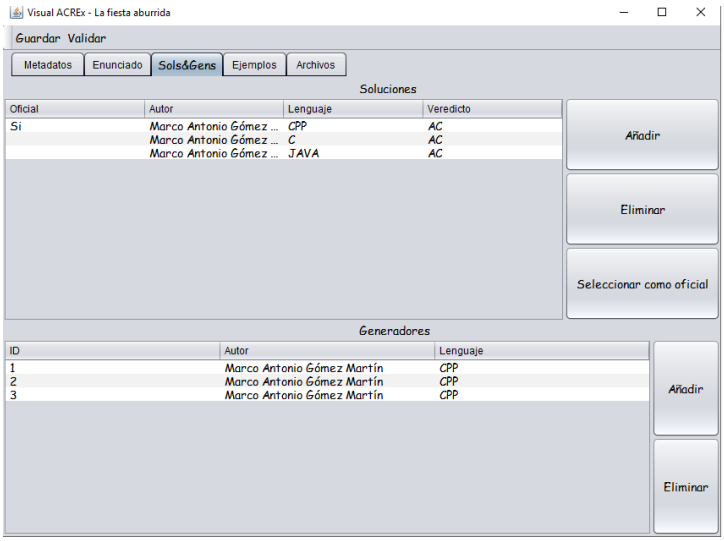


Figura 4.3: Vista de la pestaña *Soluciones y generadores*

4.2.4. Ejemplos

Esta es, junto con la pestaña de metadatos, la más sencilla de la ventana principal. Muestra el contenido de tres archivos:

- `empty.in`: incluye cómo sería una entrada vacía del problema.

- `sample.in`: entrada del caso de ejemplo, que se muestra en el enunciado del problema.
- `sample.out`: salida del caso de ejemplo, que también se muestra en el enunciado.

Los tres archivos se pueden editar, ya que se muestran en una `JTextArea` editable, y a la hora de guardar el problema se lee el texto que haya introducido el usuario.

4.2.5. Archivos

Por último, en esta pestaña se muestran todos los archivos incluidos en el directorio raíz del problema que no son referenciados en el XML o el enunciado. Esta lista se actualiza en el momento de guardar los cambios realizados, que como hemos dicho en la sección 4.2, se hace también de manera automática al validar la estructura general del problema.

Aparte de archivos que se puedan incluir de manera manual, aparecen todos los que se generan en pasos intermedios de la validación, como pueden ser los ejecutables o los resultados de compilar los códigos fuente. Además, como ya se ha mencionado en la subsección 4.2.3, aparecen los archivos de las soluciones "eliminadas".

Debajo de la lista de archivos no utilizados aparece un botón de *Eliminar*, para borrar de manera definitiva el archivo seleccionado.

4.3. Validación

4.3.1. Proceso

¿De qué serviría una herramienta para editar archivos que no nos dijera si aquello que hemos editado tiene sentido? Puede que hayamos editado el archivo del enunciado y sin querer hayamos cometido un error de formato, o que una de las soluciones que se han añadido no sea tal. Pues bien, para eso tenemos el validador, que hace una comprobación exhaustiva de todo el directorio en el que se encuentra el problema. El proceso que sigue se detiene si en algún momento una de las comprobaciones no es satisfactoria (puede que haya errores menores que no impidan la validez del problema, los cuales se notifican pero sin detener la comprobación).

Una vez que se han realizado correctamente todas las comprobaciones, se muestran los resultados del proceso. Podemos dividir los datos obtenidos en tres:

- Por un lado, se muestran aquellos avisos y pequeños errores que han podido surgir durante la validación.

- Por otro, en un diagrama de sectores aparece el número de soluciones en cada uno de los lenguajes soportados.
- Finalmente, se puede observar información referente a los generadores de casos de prueba en dos vistas distintas. Por defecto, veremos una tabla en la que aparece cada uno de los generadores en una fila, y en las columnas los datos propios de cada uno, como el consumo de memoria, el tiempo de generación y ejecución en cada una de las soluciones y valoraciones sobre la estructura. También podemos optar por ver la gráfica de los tiempos de ejecución (como se muestra en la figura 4.5) o la gráfica sobre la memoria consumida (en este caso, en la figura 4.6).

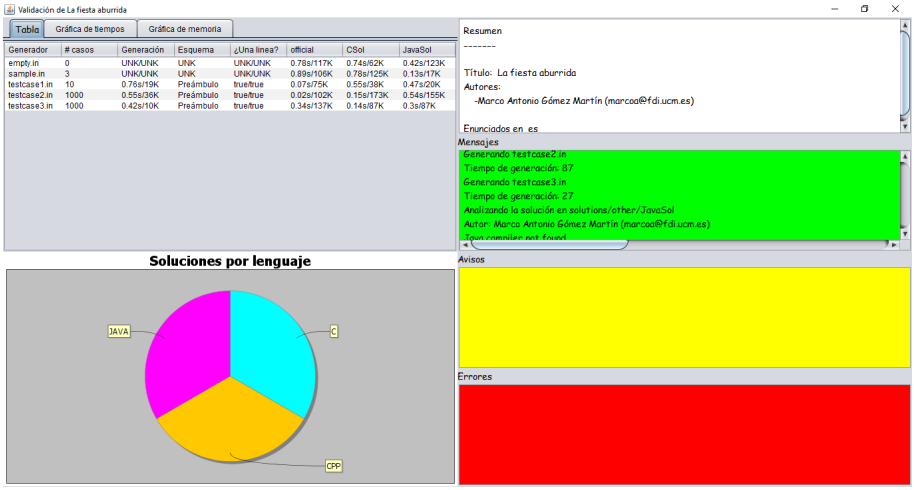


Figura 4.4: Vista, con tabla, de una validación

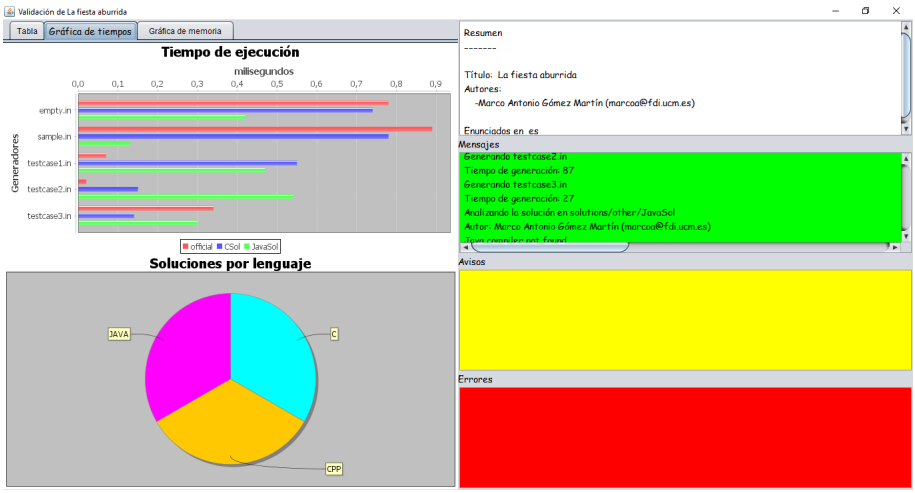


Figura 4.5: Vista, con gráfica de tiempos, de una validación

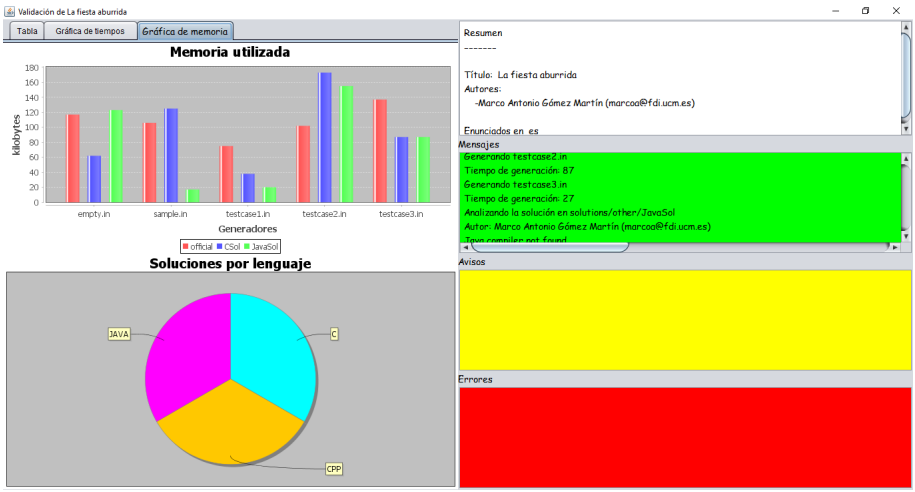


Figura 4.6: Vista, con gráfica de memoria, de una validación

Capítulo 5

Implementación

Hazlo o no lo hagas, pero no lo intentes
Star Wars: El Imperio contraataca

Resumen: en este capítulo, se detallan las diferentes decisiones de diseño que se han tomado a lo largo del desarrollo de la herramienta, así como las librerías externas que se han utilizado para que tuviese una interfaz más atractiva.

Pestañas

Como vimos en la sección 3.3, `ACREx` es un conjunto muy potente de utilidades para problemas de programación, pero las modificaciones en los mismos se hacen muy tediosas por el formato en el que se guardan. Por ello, se propone la creación de una interfaz que facilite el trabajo.

Mi primera decisión fue basar dicha interfaz en pestañas (ver figura 5.1), separando los distintos aspectos del problema para simplificar la vista. Esto permite centrar la atención del usuario en los datos importantes, y editar cada apartado de manera independiente.

Figura 5.1: Pestañas de GUI-ACREx

Guardado de `problem.xml`

Como se ha comentado en la sección 3.2, toda la información relativa al problema viene dada por el archivo `problem.xml`, así que todos los cambios que se realicen deben aparecer reflejados en el mismo.

Para editarlo, había dos opciones. Por un lado, guardar cada cambio que se realizara de manera individual, como pudiera ser añadir un autor al problema. Esto implicaba tener que programar un método específico para cada cambio potencial del problema, y en cada uno de ellos realizar el parseo de la información en nodos, atributos y árboles para buscar el campo a modificar, realizar el cambio y volver hacia arriba. Y no era muy eficiente.

Por ello, se optó por la segunda vía: los cambios que realiza el usuario se quedan en la vista, y sólo se hacen efectivos en el modelo cuando utiliza el botón de *Guardar*. En este caso, nos olvidamos de métodos individuales para cada caso de uso, ya que lo único que hay que hacer es crear un objeto de la clase `ProblemXML` (representa la información del archivo `xml` con la misma estructura que éste) y llamar a un *marshaller* para que lo vuelque al archivo. Obviamente, si por cualquier motivo el programa se cierra antes de haber guardado, todos los cambios que se hayan realizado sobre la interfaz se perderán.

Archivos huérfanos

Como se ha mencionado en la sección 4.2.5, la pestaña Archivos de la ventana principal muestra un listado con todos los archivos que no se utilizan como base del problema. Es decir, que no son código fuente de ninguna solución ni ejemplo ni generador de casos de prueba, ni es un enunciado, ni se utiliza dentro de uno de ellos. Esta lista, que es la que luego permite borrar de manera definitiva un archivo que previamente formaba parte del problema, se actualiza cada vez que se guardan los cambios, por lo que hace falta volver a confirmar que queremos borrar un archivo para que así sea. De esta manera, los cambios que se hacen efectivos son aquellos que seguro que el usuario quiere realizar.

Proceso de validación

Cuando decidimos validar un problema, se siguen una serie de pasos:

1. Comprueba que en el equipo en el que se está ejecutando hay disponibles compiladores de C, C++ y Java, que son los lenguajes soportados para el desarrollo de soluciones y generadores, así como de LaTeX, que será con lo que se generen los enunciados.

2. Verifica que hay autores para el problema, que los archivos referenciados en los enunciados se encuentran dentro del directorio raíz y que los archivos de las soluciones se encuentran en los directorios correspondientes.

3. Analiza la información de la solución oficial: debe existir, ser única y tener un resultado esperado correcto.

4. Ejecuta la solución oficial tomando como entrada el archivo .in de ejemplo, y comprueba que el resultado es el indicado en el archivo `.out` de ejemplo.

5. Lo mismo, pero en este caso con la entrada vacía, así que simplemente comprueba que la salida de la solución oficial con entrada vacía es igual a la que aparece en el archivo `empty.out`.

6. Verifica la información de los casos de prueba, al igual que del resto de soluciones (si las hubiera).

7. Comprueba que hay un y solo un enunciado de referencia, que no hay varios del mismo idioma y que no se encuentran en el mismo directorio.

La clase encargada de realizar el proceso es `Validator`, perteneciente al paquete `es.acrex.validator`. Forma parte de las utilidades originales de `ACREx`, aunque se han tenido que realizar cambios para la implementación de la interfaz. El más importante de todos es la creación de un nuevo `Output` donde mostrar el resultado de la validación. Ya que la aplicación original sólo trabajaba con consola, los mensajes de error, avisos y confirmaciones se enviaban a la salida correspondiente, pero siempre por consola. Ahora, existe un `WindowOutput`, en el que se muestra toda la información obtenida a partir de la validación.

Edición del enunciado

Como hemos visto en la subsección 4.2.2, el campo del enunciado no es editable. Esto tiene un motivo: cada usuario utiliza su editor de textos preferido (Notepad++, Emacs... o incluso el Bloc de notas nativo de Windows), y ninguna implementación de edición de texto en Java va a ser mejor que esa; por ello, se incluye el botón *Editar*, que automáticamente abre el archivo con el editor de texto por defecto del usuario.

Pestaña *Sols&Gens*

En un principio, esta pestaña estaba dividida en dos: una para soluciones y otra para generadores. Sin embargo, tras varios cambios de diseño, decidí unificarlas, ya que si no la ventana se quedaba muy vacía cuando se mostraba una de estas dos pestañas. Además, el funcionamiento de ambas es muy similar, por lo que no era tan descabellado ubicar las dos tablas en la misma pestaña.

Seaglass LookAndFeel

Las interfaces gráficas creadas con Swing no son la última moda en diseño: en general, los componentes son planos y aburridos. Por eso, decidí modificar

la apariencia, para que la aplicación tuviera algo más de cuerpo y entidad.

Java pone a nuestra disposición los *Look and Feel* (traducido a castellano, "Aspecto y sensación"), que permiten cambiar cómo se muestran los componentes gráficos diseñados con Swing. Los hay con un diseño propio, pero también otros que toman el aspecto del sistema operativo que indiquemos, ya sea Windows, Linux o Mac (en caso de éste último, solo se puede utilizar si se está ejecutando en un equipo Mac). Después de probar varios de ellos, como el *Nimbus* o el *Metal*, decidí utilizar el *Seaglass*, ya que la actualización de la vista es muy fluida y su diseño es simple pero a la vez atractivo.

Gráficas

Para mostrar las gráficas con los resultados de la validación, era necesario encontrar una herramienta que las dibujase en tiempo de ejecución, ya que es imposible tenerlas preparadas de manera fija. Por ello, se decide usar `JFreeChart`, una librería gratuita de Java centrada única y exclusivamente en mostrar gráficas. Para funcionar, necesita también que en el proyecto se encuentre la librería `JCommon`. La desarrolló David Gilbert en el año 2000, y es capaz de mostrar gráficas de todo tipo: desde diagramas de sectores, líneas o barras hasta incluso otros con forma de cuentakilómetros.

Su uso es muy sencillo, y en general se basa en crear un contenedor de datos, añadir los que queramos mostrar y crear la gráfica. El resultado es muy atractivo, y además de crear el diagrama propiamente dicho tenemos otras opciones, como incluir o no la leyenda, que se muestre más información cuando posamos el puntero del ratón encima de una de las secciones...

Capítulo 6

Conclusiones y trabajo futuro

Cuando pensamos que el día de mañana nunca llegará, ya se ha convertido en el ayer

Henry Ford

6.1. Conclusiones

En este proyecto de fin de grado se ha presentado una herramienta interactiva que facilita el trabajo de los autores a la hora de crear y modificar problemas de programación, a la vez que permite validar los problemas existentes mostrando diversos datos y estadísticas. Utiliza el formato de problemas diseñado en la Facultad de Informática, así como utilidades de `ACREx`, sistema basado en comandos de consola para realizar estas tareas.

6.2. Trabajo futuro

Como posibles mejoras de cara a la aplicación, se podrían implementar las siguientes características:

- Mayor *feed-back* de la etapa de validación, entendiendo por esto datos más pormenorizados.
- Soporte de soluciones en otros lenguajes de programación distintos a C, C++ y Java (aunque esto implica también ampliar el catálogo de lenguajes válidos para *Acepta el reto*).
- Posibilidad de subida automática de los problemas realizados al almacén de *Acepta el reto*.

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

Polygon, manual de usuario. Disponible en `http://polygon.codeforces.com`

JSON, documentación del formato. Disponible en `http://www.json.org/json-es.html`.

Kattis, wiki interna de ProblemArchive. Disponible en `http://problemarchive.com/wiki/index.php/Introduction_to_the_kattis_Problem_Format`.

JFreeChart, documentación de la librería. Disponible en `http://www.jfree.org/jfreechart/api/javadoc/index.html`

Seaglass LookAndFeel, documentación de la librería. Disponible en `http://www.jfree.org/jfreechart/api/javadoc/index.html`

–¿Qué te parece desto, Sancho? – Dijo Don Quijote –
Bien podrán los encantadores quitarme la ventura,
pero el esfuerzo y el ánimo, será imposible.

Segunda parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes

–Buena está – dijo Sancho –; fírmela vuestra merced.
–No es menester firmarla – dijo Don Quijote–,
sino solamente poner mi rúbrica.

Primera parte del Ingenioso Caballero
Don Quijote de la Mancha
Miguel de Cervantes